Low-Memory Attacks against Two-Round Even-Mansour using the 3-XOR Problem

Gaëtan Leurent, Ferdinand Sibleyras

Low-Memory Attacks against Two-Round Even-Mansour using the 3-XOR Problem

Gaëtan Leurent and Ferdinand Sibleyras

Inria, France
{gaetan.leurent,ferdinand.sibleyras}@inria.fr

Abstract. The iterated Even-Mansour construction is an elegant construction that idealizes block cipher designs such as the AES. In this work we focus on the simplest variant, the 2-round Even-Mansour construction with a single key. This is the most minimal construction that offers security beyond the birthday bound: there is a security proof up to $2^{2n/3}$ evaluations of the underlying permutations and encryption, and the best known attacks have a complexity of roughly $2^n/n$ operations.
We show that attacking this scheme with block size n is related to the 3-XOR problem with element size $\ell = 2n$, an important algorithmic problem that has been studied since the nineties. In particular the 3-XOR problem is known to require at least $2^{\ell/3}$ queries, and the best known algorithms require around $2^{\ell/2}/\ell$ operations: this roughly matches the known bounds for the 2-round Even-Mansour scheme.
Using this link we describe new attacks against the 2-round Even-Mansour scheme. In particular, we obtain the first algorithms where both the data and the memory complexity are significantly lower than 2^n. From a practical standpoint, previous works with a data and/or memory complexity close to 2^n are unlikely to be more efficient than a simple brute-force search over the key. Our best algorithm requires just λn known plaintext/ciphertext pairs, for some constant $0 < \lambda < 1$, $2^n/\lambda n$ time, and $2^{\lambda n}$ memory. For instance, with $n = 64$ and $\lambda = 1/2$, the memory requirement is practical, and we gain a factor 32 over brute-force search. We also describe an algorithm with asymptotic complexity $\mathcal{O}(2^n \ln^2 n/n^2)$, improving the previous asymptotic complexity of $\mathcal{O}(2^n/n)$, using a variant of the 3-SUM algorithm of Baran, Demaine, and Pătraşcu.

Keywords: Even-Mansour, Cryptanalysis, 3-XOR

1 Introduction

The Even-Mansour construction [12] is a very simple and elegant way to design a block cipher E from a public permutation P, defined as $E_k(x) = P(x \oplus k_1) \oplus k_2$. In the random permutation model, this construction has been proven secure as long as $D \cdot Q \leq 2^n$, with n the block size, D the data complexity (online queries to the encryption function) and Q the number of evaluation of the permutation (offline queries). In particular, the time T needed by an attacker is lower bounded by Q, therefore attacks must satisfy $D \cdot T \geq 2^n$. We also have a number of attacks

matching this bound, such as [7] with chosen plaintext or [11] using just known plaintext: when balancing online and offline queries, these attacks require only $2^{n/2}$ queries and $2^{n/2}$ computations (including all the computations required by the attack, in addition to permutation queries). A single-key version of the Even-Mansour construction has also been proposed with the same security [10], defined as $E_k(x) = P(x \oplus k) \oplus k$.

More recently, this construction was generalized to the iterated Even-Mansour scheme, also called key-alternating cipher [3]. The r-round construction uses r independent permutations and $r+1$ keys, and can be considered as an idealization of concrete SPN ciphers:

$$E_k(x) = P_r\Big(\cdots P_2\big(P_1(x \oplus k_0) \oplus k_1\big)\cdots\Big) \oplus k_r$$

This construction was first proven to be secure up to $2^{2n/3}$ queries for $r \geq 2$ [3], and later improved to $2^{nr/(r+1)}$ queries [17,6].

As in the single-round case, the requirement to have independent keys and independent permutations can be relaxed without reducing the security. In particular, two single-key variants of the 2-round Even-Mansour have been proposed [5]:

$$\begin{aligned}
&\text{EMIP}: \quad E_k(x) = P_2\big(P_1(x \oplus k) \oplus k\big) \oplus k \\
&\text{EMSP}: \quad E_k(x) = P\big(P(x \oplus k) \oplus \pi(k)\big) \oplus k, \quad \text{with } \pi \text{ a linear orthomorphism.}
\end{aligned}$$

The EMIP construction uses two independent permutations, while the EMSP construction uses a single permutation, and a fixed linear orthomorphism (a linear operation such that both $x \mapsto \pi(x)$ and $x \mapsto x \oplus \pi(x)$ are invertible, such as multiplication by a constant in a field).

There are simple key-recovery attacks matching the $2^{nr/(r+1)}$ bound on the number of queries given in [3], but even with $r = 2$ the best known attacks require about $2^n/n$ operations (in addition to the queries). Attacks against the 3-round Even-Mansour construction have also been given in [8], with complexity close to $2^n/n$, and no attack better than 2^n is known for $r > 3$.

In this paper we focus on the most simple instances, the 2-round variants of EMIP and EMSP, collectively denoted as 2EM, and we look for better attacks than what is currently known, with a focus on low memory and low data.

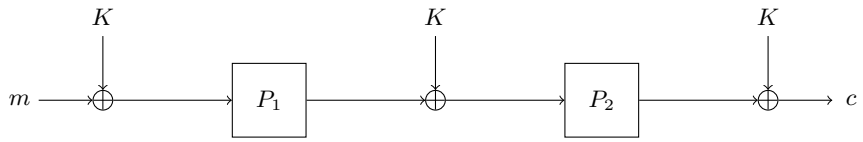

Fig. 1. Single key two-round Even-Mansour scheme (2EM) EMIP variant

Previous works. The first non-trivial attack against an iterated Even-Mansour construction was described by Nikolic, Wang, and Wu in [19] against the two-round EMIP construction $P_2\big(P_1(x \oplus k) \oplus k\big) \oplus k$, using multi-collisions. The main idea is to consider the function $\phi : u \mapsto P_1(u) \oplus u$, and to evaluate it on a large number of points, so as to identify a particular value v that occurs more frequently than others (at least t times). Then, for each known plaintext pair $\big(x, E(x)\big)$, the attacker assumes that $\phi(x \oplus k) = v$, *i.e.* $P_1(x \oplus k) \oplus k = x \oplus v$; this gives a key candidate $P_2(x \oplus v) \oplus E(x)$. Since the assumption holds for at least t values of x, the expected complexity is $2^n/t$.

According to the asymptotic analysis performed in [18], the optimal choice is to set $t = \Theta(n/\ln n)$. A value with this number of repetitions is expected after evaluating ϕ roughly $2^n/n$ times, so that the total complexity of this attack is $2^n \ln n/n$, asymptotically smaller than 2^n.

This attack was later improved by Dinur, Dunkelman, Keller and Shamir [8]. In particular, they describe a variant with lower online complexity using N_v different values v_i that appear t times each, with a smaller value of t. Each online pair $\big(x, E(x)\big)$ is then used to make a key guess with every v_i, which reduces the data complexity to $2^n/N_v t$. They didn't evaluate this strategy asymptotically, but they computed that $N_v = 2^n \mu^t e^{-t}/t!$ multi-collisions should be found, when evaluating a fraction μ of the domain. In particular, with $\mu = 1/n$ and $t = o(n/\ln n)$, we have an upper bound on the data complexity: $2^n/N_v \leq n^{2t} = \exp(2t \ln n)$, which is asymptotically smaller than $2^{\lambda n}$ for any $\lambda > 0$. The time complexity is still $2^n/t$. Variants of the attack that can applied to Even-Mansour schemes with a linear key-schedule, such as EMSP are also given in [9].

Dinur *et al.* also proposed attacks against a more general construction with 3 independent keys, using multi-collisions to find differential properties of the random permutation. However this attack only reaches time complexity $\mathcal{O}(2^n/\sqrt{n/\ln n})$.

All those attacks require a large pre-processing step to discover multi-collisions: a t-collision is only expected after $2^{n(t-1)/t}$ evaluations of ϕ. Moreover, the best known algorithm to locate multi-collisions requires a memory of size $2^{n(t-2)/t}$ [16]. Therefore, multi-collision based techniques intrinsically require time and memory close to 2^n (asymptotically, we need to have t approaching infinity in order to gain a non-constant advantage over brute-force attacks).

In the journal version of their paper, Dinur *et al.* show an interesting side-result on EMIP. They describe an alternative attack with low memory using linear algebra [9, Section 4.2]. In this attack, they evaluate $\phi : u \mapsto P_1(u) \oplus u$ on a small set of λn values ($0 < \lambda < 1/3$), and they look for linear relations that are satisfied by all $\phi(u)$ in the set: $L(\phi(u)) = 0$ with $n - \lambda n$ equations. Then, for a given plaintext pair $\big(x, E(x)\big)$, if $x \oplus k$ is in the set, this implies linear relations on $z = k \oplus P_1(x \oplus k)$, the input of P_2: $L(z) = L(x)$. Finally, using structures for x and z, a match can be identified using linear relations on the key (following from the assumption that $x \oplus k$ is in the set), using $k = P_2(z) \oplus E(x)$. The full details of the attack are given in [9]. This attack only requires a memory of size $2^{\lambda n}$ to store the structures, but it requires $2^n/\lambda n$ chosen plaintext pairs. However, this

approach is not applicable to 3EM or 2EM with independent keys, which are the main focus of their work.

More recently, Isobe and Shibutani [14] introduced Meet-in-the-Middle techniques to attack the 2-round Even-Mansour construction. The basic variant of their attack uses a function f depending on a bits of the key k_f (with a in the order of $\ln n$), and a function g depending on the remaining $n-a$ bits k_g. Furthermore, they use a starting point such that a output bits of f are actually independent of the key k_f. This allows them to do the matching over P_2 using just k_g. The attack requires time and data 2^{n-a}, with chosen plaintexts.

The function f is such that it is equivalent to looking for partial multi-collisions in ϕ while imposing a structure on the inputs: they fix $n-a$ bits of u and hope that a outputs bits of $\phi(u)$ will be independent of the remaining a bits of u. For this to work the parameter a must satisfy $a \cdot (2^a - 1) \leq n - a$, and Isobe and Shibutani only give concrete parameters for some values of n. Asymptotically, the maximal value of a can be found by solving $a \cdot (2^a - 1) = n - a$; since $a \lll n$ and $1 \lll 2^a$, we have $a \approx W(n \ln 2)/\ln 2 \approx \log n - \log\log n$, using the Lambert W function.

They also describe a low-data complexity variant of the attack, where the starting point is dynamically chosen so that $a + d$ bits of the plaintext are fixed. This reduces the data complexity to 2^{n-d-a}, while the time complexity is still 2^{n-a}. The parameters are more constrained and must satisfy $a \cdot 2^a + d \leq n - a$. If we want to achieve a data complexity of $2^{\lambda n}$ for a constant $0 < \lambda < 1$, we can set $d = n - \lambda n$, and $a = \log\lambda + \log n - \log\log n$. This gives a time complexity of $2^n \log n / \lambda n$.

Finally, they give a time-optimized attack where $b = a + c$ output bits of f are independent of k_f (instead of just a). This reduces the number of queries and memory needed for the matching to 2^{n-b}, but the attack still requires 2^{n-a} memory accesses and chosen plaintext. The parameters must satisfy $b \cdot 2^a + b - a \leq n - b$, but the authors only give concrete values for some choices of n, and no asymptotic analysis. However, we can observe that we must have $b \cdot 2^a \leq n$; in particular, if we want an attack with an advantage that is not asymptotically bounded, we need to have a approaching infinity and therefore b/n approaching zero (this attack cannot reduce the memory to $2^{\lambda n}$ with $\lambda < 1$). In particular, the optimal parameters satisfy $b \cdot 2^a + b - a = n - b$, with $b \lll n$ and $a \lll 2^a$, hence $b \cdot 2^a \approx n$. Therefore we have a complexity of roughly 2^{n-b} in queries and memory, and $b2^n/n$ in time and data, with $\log n \leq b \lll n$.

All those attacks are summarized in Tables 1 and 2. We point out that the complexity reported in [14] is lower than listed here, because the authors assume that a memory access to a large table is significantly cheaper than the evaluation of the public permutations P_i. Given that a public permutation can obviously be implemented with a table lookup if memory is fast and cheap, we assume that a memory access to a table of size roughly 2^n cannot be faster than the evaluation of the P_i permutations.

Table 1. Comparison of attacks against 2EM. Asymptotic complexity, up to constants. "Data" denotes encryption queries, while "Queries" denotes calls to the public permutations P_i.
$0 < \lambda < 1$; $\log n \leq \beta \lll n$; KP: Known plaintext; CP: Chosen plaintext.

Ref	Data		Queries	Time	Memory	Comment
[19]	$2^n \ln n/n$	KP	$2^n \ln n/n$	$2^n \ln n/n$	$2^n \ln n/n$	Multi-collisions
[8]	$2^n \sqrt{\ln n/n}$	CP	$2^n \sqrt{\ln n/n}$	$2^n \sqrt{\ln n/n}$	$2^n \sqrt{\ln n/n}$	Diff. m-c (indep. keys)
[8]	$2^{\lambda n}$	KP	$2^n \ln n/n$	$2^n \ln n/n$	$2^n \ln n/n$	Multi-collisions
[9]	$2^n/\lambda n$	CP	$2^n/\lambda n$	$2^n/\lambda n$	$2^{\lambda n}$	Linear algebra
[14]	$2^n \ln n/n$	CP	$2^n \ln n/n$	$2^n \ln n/n$	$2^n \ln n/n$	MitM
	$2^{\lambda n}$	CP	$2^n \ln n/n$	$2^n \ln n/n$	$2^n \ln n/n$	MitM
	$2^n \beta/n$	CP	$2^n/2^\beta$	$2^n \beta/n$	$2^n/2^\beta$	MitM
S. 3.3	n	KP	$2^n/\sqrt{n}$	$2^n/\sqrt{n}$	$2^n/\sqrt{n}$	3XOR [15]
S. 4.1	2^d	KP	$2^{n-d/2}$	$2^n/n$	$2^{n-d/2}$	Clamping + 3XOR [4]
S. 4.3	2^d	KP	$2^{n-d/2}$	$2^n \ln^2 n/n^2$	$2^{n-d/2}$	Clamping + 3XOR [1]
S. 4.4	λn	KP	$2^n/\lambda n$	$2^n/\lambda n$	$2^{\lambda n}$	Low Data Filter

Our results. The main results of the paper are the three key-recovery attacks on EMIP given in Section 4 whose complexities are summarized in Tables 1 and 2. To the best of our knowledge these are the first attacks on EMIP to significantly reduce simultaneously the data and the memory complexities below 2^n. The first attack, Section 4.1, shows that we can achieve the best computational time complexity known so far, that is $\mathcal{O}(2^n/n)$, while using just as much data and queries as the best known distinguisher which is optimal in the balanced case ($2^{2n/3}$ calls to E, P_1 and P_2) with a memory usage not exceeding the number of queries. The next attack in Section 4.3 works exactly the same way only it is using another generic 3-XOR algorithm which improves the asymptotic time complexity to $\mathcal{O}(2^n \ln^2 n/n^2)$ that beats the best one known so far. However this 3-XOR algorithm is believed to be impractical for realistic block sizes, notably for $n = 64$. And the third attack in Section 4.4 uses very low data, λn, and possibly low memory, $2^{\lambda n}$, for some $\lambda < 1$ while keeping a competitive asymptotic time complexity of $\mathcal{O}(2^n/\lambda n)$.

We also present some security reduction notably showing that adding a linear key schedule does not protect against generic attacks on EMIP. This effectively extends the scope of our attacks in particular showing they can also be applied to the EMSP variant. We also explain the link between the 3-XOR problem and the key-recovery attacks on EMIP showing how one can help us solve the other which justifies our approach. Then we exhibit a symmetry in the Even-Mansour construction that shows how, in the chosen ciphertext attack (CPA) model, an attacker can always swap the number of queries he is making to E, P_1 and P_2 to optimize on the most available resources. This implicitly extends these and previous attacks to adapt to many different data and query complexity profiles.

Table 2. Comparison of attacks against 2EM with $n = 64$. The complexity unit is one evaluation of the cipher; we assume that computing P_1 or P_2 costs $1/2$, and that a memory access to a large table also costs $1/2$. The time complexity also includes the time necessary to generate the data.

Ref	Data		Queries	Time	Memory	Comment
[19]	$2^{58.7}$	KP	$2^{60.5}$	$2^{60.9}$	2^{60}	Multi-collisions
[8]	2^{45}	KP	$2^{60.7}$	$2^{60.7}$	2^{60}	Multi-collisions
[9]	2^{60}	CP	2^{59}	$2^{60.6}$	2^{16}	Linear algebra
[14]	2^{60}	CP	2^{60}	$2^{61.3}$	2^{60}	MitM
	2^{8}	CP	2^{62}	$2^{62.6}$	2^{62}	MitM
	2^{61}	CP	2^{57}	$2^{61.7}$	2^{58}	MitM
Sec. 3.3	2^{6}	KP	2^{61}	2^{62}	2^{61}	3XOR
Sec. 4.1	2^{42}	KP	2^{43}	2^{58}	2^{42}	Clamping + 3XOR [4], bal. case
	2^{14}	KP	2^{57}	$2^{58.6}$	2^{57}	optim. data
Sec. 4.2	2^{35}	CP	2^{57}	$2^{58.6}$	2^{35}	optim. memory & swap $E \leftrightarrow P_1$
Sec. 4.3	2^{42}	KP	2^{43}	N.A.	N.A.	Clamping + 3XOR [1], bal. case
Sec. 4.4	2^{5}	KP	2^{59}	2^{60}	2^{32}	Low Data Filter $\lambda = 1/2$
	2^{4}	KP	2^{60}	2^{61}	2^{16}	$\lambda = 1/4$

Lastly we generalize our approach to show that a single key r rounds Even-Mansour scheme can be rewritten as a structured $(r+1)$-XOR problem with words of size rn. Interestingly both the single key r rounds Even-Mansour and the $(r+1)$-XOR problem with words of size rn have a simple information theoretic solver using $2^{\frac{r \cdot n}{r+1}}$ queries though solving these uses more computations than a brute-force solution for $r \geq 4$.

Practical considerations. In a practical setting, the data complexity and the memory complexity are important considerations. In particular, an attack with complexity $2^n/n$ is unlikely to be more efficient than a brute-force attack if it requires almost 2^n data, or almost 2^n memory. As mentioned above, some of the previous attacks can reduce the data complexity to $2^{\lambda n}$ for an arbitrary $\lambda > 0$, and the attack from [9, Section 4.2] can reduce the memory to $2^{\lambda n}$, but so far none of them can simultaneously reduce the data and memory complexity below $2^{\lambda n}$ for $\lambda < 1$.

Besides, multi-collision based attacks can use a sequential memory (such as a hard drive) and sort values to locate collisions while the Meet-in-the-Middle attacks require random access memory, with $\Theta(2^n \ln n/n)$ accesses to a table of size $\Theta(2^n \ln n/n)$.

On the other hand the linear algebra techniques we use in our attacks will require algorithmic tricks very close to what was done by Bouillaguet, Delaplace and Fouque [4] for the 3-XOR problem. In particular the values we deal with are sufficiently random to be sorted linearly and the right matrix multiplication in GF(2) LM for an exponentially large matrix L can be computed with a number of operations linear in the size of L. Many constant time optimizations are therefore

omitted in this work which justify that right multiplications, sorting and merging two big lists L_1 and L_2 take time and space $\mathcal{O}(|L_1| + |L_2|)$. This is consistent with previous cryptanalysis on EMIP.

For the cost of queries to the oracles E, P_1 and P_2 we mainly follow the convention established by Dinur *et al.* [9] which states that an online query to E costs 1 unit of computation implying that P_1 and P_2 cost 1/2. The main advantage is that it makes it easy to compare with the brute-force solution that would use 2^n computations. The disadvantage is that it makes it hard to combine with the computations used for simple operations: an evaluation of a cryptographically secure permutation should cost more than a XOR operation.

We give concrete complexity values for $n = 64$ in Table 2 with the assumption that a combination of some linear time operations does not exceed the cost of computing a permutation that is 1/2 time unit. Concretely, iteratively right multiplying, sorting and merging two lists L_1, L_2 costs $|L_1|/2+|L_2|/2$. We believe this makes an honest comparison with previous works though they may use other assumptions.

Organization of the paper. First, in Section 2, we show some reductions that extend our results and justify our approach. Then in Section 3 we take a close look on previous works done on the 3-XOR Problem to show how it can help the cryptanalysis of EMIP. Lastly, in Section 4, we devise three dedicated algorithms for EMIP each having their own particular complexity trade-off. Also we extend our approach in Appendix 5 to the r rounds iterated Even-Mansour construction.

Notations. We denote the block size of the Even-Mansour scheme (*i.e.* the width of the public permutations) as n, and the concatenation of n-bit blocks x and y as $x \parallel y$. When x and y fit together in one block, we use $x|y$ to denote their concatenation. We use $L[i]$ to denote element i of list L, $x_{[i]}$ to denote bit i of x, $x_{[i:j]}$ to denote bits i to $j-1$, 0 to denote a zero GF(2) matrix and I to denote an identity GF(2) matrix. When L is a list of ℓ n-bit values, we identify it with a $\ell \times n$ matrix where the elements of L are the rows of the matrix. Finally, we use a curly brace for systems of equations.

2 Security Reductions

We start with some general observations about the security of iterated Even-Mansour schemes. In particular, we show that we can focus on the EMIP construction without loss of generality, how to reduce the security of this construction to an instance of the 3-XOR problem, and how to reorder the oracles to achieve many different trade-offs.

Some previous works already implicitly took advantage of such reductions. For example Isobe and Shibutani [14] realised that their recent attack on EMIP is also applicable to EMSP and Dinur *et al.* [9] realised that they could reorder the oracles for their cryptanalysis of reduced round LED. We formally show here that these tricks are in fact real security reductions and do not depend on the approach used.

2.1 Removing the Key Schedule

There are several variants of single-key multiple-round Even-Mansour studied in the literature. The most general form uses two independent permutations, and an arbitrary key schedule (see Figure 2):

$$E_k(x) = P_2\big(P_1(x \oplus \gamma_0(k)) \oplus \gamma_1(k)\big) \oplus \gamma_2(k).$$

According to the analysis of [5], there is a class of good key schedules where the γ_i's are public linear bijective functions. In the following, we focus on this class of key schedules, *i.e.* we assume that the $\gamma_i \in \mathsf{GL}(\mathbb{F}_2^n)$. In order to simplify the analysis, we reduce the security of this construction to the security of the EMIP variant without a key schedule.

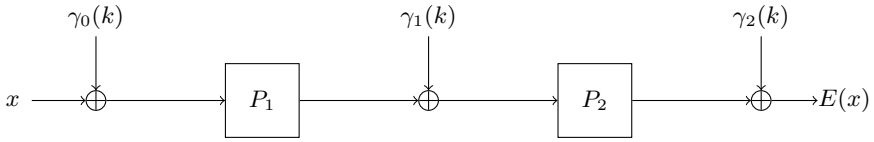

Fig. 2. Linear key-schedule 2-round Even-Mansour.

The main trick is to rewrite the addition of the subkey $\gamma_i(k)$ as the application of the inverse γ_i^{-1}, the addition of k and the application of the forward γ_i:

$$\begin{aligned} x \oplus \gamma_i(k) &= \gamma_i\big(\gamma_i^{-1}(x \oplus \gamma_i(k))\big) \\ &= \gamma_i\big(\gamma_i^{-1}(x) \oplus k\big) \end{aligned}$$

which works thanks to γ_i being linear. Then we define E', P_1', P_2' as follows:

$$P_1'(x) = \gamma_1^{-1}\big(P_1(\gamma_0(x))\big) \quad P_2'(x) = \gamma_2^{-1}\big(P_2(\gamma_1(x))\big) \quad E'(x) = \gamma_2^{-1}\big(E(\gamma_0(x))\big)$$

Thanks to the previous relation, E', P_1', P_2' is actually an instance of EMIP with the same key k (see Figure 3):

$$E'(x) = P_2'\big(P_1'(x \oplus k) \oplus k\big) \oplus k.$$

Therefore, any attack against EMIP can be used on E', P_1', P_2', and break the initial construction with a key schedule. In particular, a key-recovery attack against EMIP will recover the key of the more general scheme of 2EM.

In the following we only consider the EMIP variant without a key schedule, but thanks to this reduction our attacks can be applied to many other 2EM variants, including the EMSP construction of [5].

Definition 1 (EMIP key recovery). *Given oracle access to three permutations E, P_1, P_2 and their inverses, with the promise that there exist k such that $E(x) = P_2\big(P_1(x \oplus k) \oplus k\big) \oplus k$, recover k.*

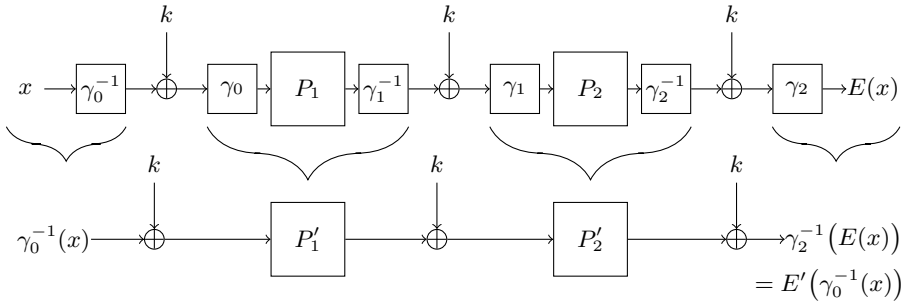

Fig. 3. Reduction of linear key schedule 2EM to EMIP.

2.2 Reduction to 3-XOR

Instead of directly focusing on a key-recovery attack, we focus on locating a triplet of values x, y, z such that the encryption of x is evaluated with permutation call $P_1(y)$ and $P_2(z)$. Formally, we say that x, y, z is a *right* triplet when $y = x \oplus k$ and $z = P_1(y) \oplus k$. A right triplet corresponds to a sequence of intermediate values in the Even-Mansour encryption, as shown in Figure 4: $\big(x, y = x \oplus k, P_1(y), z = P_1(y) \oplus k, P_2(z), E(x) = P_2(z) \oplus k\big)$; we call this sequence a *path*.

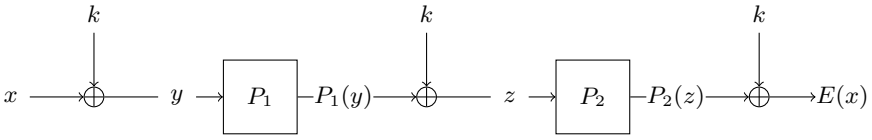

Fig. 4. A right triplet gives a path of EMIP

Since the permutations P_1 and P_2 are public, it is easy to compute a path given the key. Recovering the key from a path is also easy (we have $k = x \oplus y$), but it is hard to identify a right triplet corresponding to a path without the key. By definition a triplet is right when it follows the relation $\mathcal{R}$ defined as:

$$\mathcal{R}(x, y, z) := \begin{cases} x \oplus y = k \\ P_1(y) \oplus z = k \\ P_2(z) \oplus E(x) = k \end{cases} \tag{1}$$

$$\Rightarrow \begin{cases} x \oplus y = P_1(y) \oplus z \\ x \oplus y = P_2(z) \oplus E(x) \end{cases} \tag{2}$$

Notice that we can't directly observe (1) since we don't know k but we can easily verify the implied relation (2).

We claim that if one takes a random triplet combination and observes that it respects (2), then it is a right triplet with good probability. Indeed there are 2^n possible paths (one for every possible input x) implying as many right triplets and 2^{3n} possible triplet combinations; thus a random triplet will be right with probability 2^{-2n}. Since (2) is a $2n$-bit relation, a random but false triplet respects (2) also with probability 2^{-2n}. Therefore we can expect roughly as many right triplets than false triplets that respect (2), thus the first one we find is right with probability $\Omega(1)$. So from now on and for simplicity we will focus on filtering and recovering a triplet that simply respects (2). This means that our algorithms fails to recover the key on some instances, but they have a constant (non-zero) probability of success. In order to improve the success probability arbitrarily close to one, it is easy to test the triplets, and continue the attack until we find a right triplet (alternatively, the whole attack can just be repeated).

In order to simplify the analysis, the condition (2) can be rewritten as:

$$\begin{cases} \big(x \qquad\quad\big) \oplus \big(y \oplus P_1(y)\big) \oplus \big(z \qquad\big) = 0 \\ \big(x \oplus E(x)\big) \oplus \big(y \qquad\quad\big) \oplus \big(P_2(z)\big) = 0 \end{cases}$$

Therefore, finding a triplet satisfying (2) is equivalent to solving an instance of the 3-XOR problem, defined as:

$$\begin{aligned} f_0(x) &:= x \qquad\quad \| \; x \oplus E(x) \\ f_1(y) &:= y \oplus P_1(y) \| \; y \\ f_2(z) &:= z \qquad\quad \| \; P_2(z) \end{aligned} \tag{3}$$

The 3-XOR Problem is a well known algorithmic problem; it is a special case of k-XOR problem analyzed by Wagner as the generalized birhtday problem [20].

Definition 2 (3-XOR problem). *Given three functions f_0, f_1, f_2, find three inputs (x_0, x_1, x_2) such that $f_0(x_0) \oplus f_1(x_1) \oplus f_2(x_2) = 0$.*

We usually focus on functions f_0, f_1, f_2 that are chosen at random. Equivalently, we can be given lists L_0, L_1, L_2 (of random elements) instead of functions. The presentation with functions makes it more clear that the adversary can choose how many queries he makes to each of the functions.

EMIP Key Recovery from the 3-XOR Problem. From the previous discussion, solving the 3-XOR instance defined by (3) gives a triplet satisfying $\mathcal{R}$, which has a high probability of being a right triplet and revealing the key. Evaluating each of the f_i functions requires a single computation of a permutation. However evaluating f_0 must be done online (using an oracle call to E) because it depends on the key, while evaluating f_1 and f_2 can be done offline as the permutations are public and computable at will by the attacker. As per our adopted convention, an evaluation of f_0 —that is a call to E —costs 1 unit of computation and an evaluation of f_1 or f_2 costs 1/2.

We denote the list of values of f_i evaluated by an attacker as L_i. Therefore, the data complexity of an attack is equal to $D = |L_0|$. The time complexity is the

amount of computation required to break the scheme. In the computational model, it will depend on the algorithm used and be denoted as T. In the information theoretic model we only look at the number of calls to the permutations and denote it Q, with $Q = (|L_1| + |L_2|)/2$. We will discuss both models.

As seen from the description in (3), we can choose some parts of the values in L_i. However, if we only use random values of x, y, z to build the lists, we obtain a random 3-XOR instance with words of size $w = 2n$. It is known that to find a solution of a 3-XOR problem with good probability, the lists size should respect $|L_0| \times |L_1| \times |L_2| \geq 2^w$. In the information theoretic setting this gives a key recovery attack with $D \times Q^2 = 2^{2n}$. This is the exact same complexity trade-off as the information theoretic distinguisher described by Gaži [13]. In particular it is known that this trade-off is proven optimal in the balanced case $D = Q = 2^{2n/3}$ [5].

2.3 Symmetry between E, P_1 and P_2

In the 3-XOR problem the 3 functions behave essentially in the same way; if one has a solver using a few evaluations f_0 and lots of evaluations of f_1 and f_2, then the same solver could decide to use lots of queries to f_0 and f_1 and use fewer f_2 queries (just by permuting the functions). In our case, a natural choice is to minimize the number of evaluations of f_0, because they correspond to online queries. This ensures that we have $D \leq Q$. While this is easy to do with a 3-XOR approach, it is not obvious whether this can be done in general for an Iterated Even-Mansour key recovery. We now show that in the chosen ciphertext setting an attacker can actually permute the functions E, P_1 and P_2, and minimize the amount of online queries.

We assume that we are given an instance E, P_1, P_2 of EMIP, *i.e.* we have oracle access to E, P_1, P_2 denoting forward computations of the permutations, and E^{-1}, P_1^{-1}, P_2^{-1} denoting backward computations. We use a black-box solver $\mathcal{S}(E, E^{-1}, P_1, P_1^{-1}, P_2, P_2^{-1})$ that uses α calls to E/E^{-1} (online queries), β calls to P_1/P_1^{-1} and γ calls to P_2/P_2^{-1} and outputs the key k.

The trick is that we can rewrite the EMIP instance E, P_1, P_2, by permuting the oracles. For instance we have $P_1(x) = k \oplus P_2^{-1}(k \oplus E(k \oplus x))$ (directly from the definition of E), which gives the following EMIP instance with the same secret key k:

$$E' = P_1 \qquad\qquad P_1' = E \qquad\qquad P_2' = P_2^{-1}.$$

Therefore, we can use the solver as $\mathcal{S}(P_1, P_1^{-1}, E, E^{-1}, P_2^{-1}, P_2)$ to recover k using β online queries. Similarly, we can write $P_2(x) = k \oplus E(k \oplus P_1^{-1}(k \oplus x))$; therefore, we can use the solver as $\mathcal{S}(P_2, P_2^{-1}, P_1^{-1}, P_1, E, E^{-1})$ to recover k using γ online queries.

We could further use E^{-1} to rewrite P_1^{-1} and P_2^{-1} in the same fashion and obtain all the possible trade-off between α, β and γ. The point is that, given any solver $\mathcal{S}$, it is always up to the attacker to choose what is the most accessible data.

From here onward all of our discussed trade-off will have $|L_0| \leq \min(|L_1|, |L_2|)$ to lower the query complexity but one can remember it is an arbitrary choice.

In particular, this trick can be applied to the attack of [9, Section 4.2]. Indeed, this attack uses λn queries to P_1, with $0 < \lambda < 1/3$ and $2^n/\lambda n$ queries to E and P_2. Using this trick we can reduce the data complexity from $2^n/\lambda n$ to λn, without affecting the other parameters. Actually, the attack presented in Section 4.4 can be seen as an improved variant of this modified attack (using known plaintext rather than chosen plaintext).

3 2EM Attacks from 3-XOR Algorithms

In this section we explore the link between 2EM key recovery and the 3-XOR problem. First, we review existing approaches to solve the 3-XOR problem, and we show that previous 2EM attacks can be reinterpreted in a 3-XOR framework. Then we describe new attacks against 2EM based on the reduction of the previous Section. In this section, we focus on a generic 3-XOR instance given by three w-bit function f_0, f_1 and f_2, or three lists L_0, L_1, L_2.

3.1 3-XOR Algorithms

The Birthday Problem, that is the problem of finding collisions among two lists, has been well studied and proven useful in a number of cryptanalysis. In 2002, Wagner proposed a natural extension of this problem, the Generalized Birthday Problem [20], that is the problem of finding collisions among k lists. Here we refer to this problem as the k-XOR problem. In particular Wagner left the hard case of $k = 3$ as an open problem. His best algorithm would just take one value of the first function and solve the classical Birthday Problem among the two others, with complexity $2^{w/2}$.

Subsequent works tried to address this open problem. Two main approaches managed to improve the time complexity of the 3-XOR: an approach based on partial multi-collisions by Nikolic and Sasaki [18] and an approach using linear algebra by Joux [15]. Unfortunately, those two solutions seem hard to combine.

Multi-collisions Algorithms. Nikolic and Sasaki [18] introduced a multi-collision algorithm for the 3-XOR problem as follows. First, compute many outputs of f_0 and look for the most frequent $w/2$-bit prefix α appearing. Store all the values with this fixed prefix in a list L_0 (a partial multi-collision for f_0). Then evaluate f_1 and f_2, $2^{w/2}/\sqrt{|L_0|}$ times each, and store the results in lists L_1 and L_2. Sort the lists, and look for pairs with a difference α in the first $w/2$ bits. An average, there should be $2^{w/2}/|L_0|$ such pairs, and there is a high probability that one of them sums to a value in L_0. According to their analysis, the optimal attack uses around $2^{w/2}/w$ evaluations of f_0, resulting in a multi-collision of size $\Theta(w/\ln(w))$; therefore this algorithm solves the 3-XOR problem with complexity $\mathcal{O}\big(2^{w/2}/\sqrt{w/\ln(w)}\big)$.

Linear algebra. The second approach, introduced by Joux [15], uses linear algebra and reaches a slightly better complexity of $\mathcal{O}(2^{w/2}/\sqrt{w})$. This attack uses just $w/2$ evaluations of f_0 stored in a list L_0, and $2^{w/2}/\sqrt{w/2}$ evaluations of f_1 (resp. f_2) stored in a list L_1 (resp. L_2). Instead of collecting values in L_0 with a common prefix, we use Gaussian reduction to find a non-singular matrix M such that the elements of $L_0 \cdot M$ start with $w/2$ zeroes.[1] Then we focus on a modified 3-XOR instance:

$$L'_0 = L_0 \cdot M \qquad L'_1 = L_1 \cdot M \qquad L'_2 = L_2 \cdot M.$$

The new instance has the same solutions ($L'_0[h] \oplus L'_1[i] \oplus L'_2[j] = 0 \Leftrightarrow L_0[h] \oplus L_1[i] \oplus L_2[j] = 0$), but the elements of L_0 start with $w/2$ zeroes. Therefore, as in the previous attack, we can efficiently find the solution after sorting the lists L_1 and L_2.

This approach was later generalized by Bouillaguet, Delaplace and Fouque [4], in order to deal with instances of the 3-XOR problem where the size of the lists is limited: given three lists with $|L_0| \cdot |L_1| \cdot |L_2| = 2^w$, they solve the 3-XOR problem with complexity $\mathcal{O}(|L_0| \cdot (|L_1| + |L_2|)/w)$. In particular, with three lists of size $2^{w/3}$ this gives a time complexity of $\mathcal{O}(2^{2w/3}/w)$.

In addition, this algorithm can be combined with the clamping trick of Bernstein to reduce the memory: the attacker first filters the lists L_i to keep only values that start with $w/4$ zero bits, and solves a shorter 3-XOR instance on $3w/4$ bits. If the initial lists have $2^{w/2}$ elements, the filtered lists still have $2^{w/4}$ elements, which is sufficient to expect a solution. This gives an algorithm with time $\mathcal{O}(2^{w/2})$ and memory only $\mathcal{O}(2^{w/3})$. Arguably, this is more practical that algorithms using $\mathcal{O}(2^{w/2}/w)$ memory.

BDP Algorithm. Even before these two approaches, Baran, Demaine and Pătraşcu [1] proposed an algorithm for the 3-SUM problem (using modular additions instead of XORs) with the asymptotical complexity of $\mathcal{O}(2^{w/2} \cdot \ln^2(w)/w^2)$. This algorithm has been adapted to the 3-XOR problem by Bouillaguet *et al.* [4] with the same complexity. This is best known asymptotic complexity for the 3-XOR problem, even though the algorithm is highly impracticable for realistic values of w. We nevertheless use this algorithm to cryptanalyse 2EM in Section 4.3.

3.2 Revisiting Previous Cryptanalysis

Interestingly all attacks so far on 2EM use the same techniques as developed against the 3-XOR problem. Most of the attack are based on multi-collisions [19,8,9], and the MitM attack by Isobe and Shibutani [14] can also be interpreted as looking for a structured partial multi-collision, as seen in Section 1. On the other hand, the attack from [9, Section 4.2] uses linear algebra.

[1] For instance, we write L_0 as a block matrix $\begin{bmatrix} A & B \end{bmatrix}$ with two $w/2 \times w/2$ sub-matrices. If B is non-singular, we can use $M = \begin{bmatrix} I & 0 \\ B^{-1}A & B^{-1} \end{bmatrix}$

Using the Reduction to 3-XOR. As explained in Section 2.2, we can use an attack against 3-XOR to build a key-recovery against 2EM in a generic way. In particular, this reduction gives attacks similar to the known attacks on 2EM if we start from multi-collision algorithms to solve 3-XOR. More precisely, the reduction leads to a 3-XOR instance with $w = 2n$, defined as:

$$\begin{aligned} f_0(x) &:= x && \| \; x \oplus E(x) \\ f_1(y) &:= y \oplus P_1(y) \| \; y \\ f_2(z) &:= z && \| \; P_2(z) \end{aligned} \tag{3}$$

If we directly apply the previous algorithm the time complexity will be $\mathcal{O}\big(2^n/\sqrt{n/\ln(n)}\big)$. Concretely the most natural way would be to search for prefix multi-collisions offline in f_1 as it is computationally intensive. Because of the definition of f_1, the second half y won't repeat but $(y \oplus P_1(y))$ should repeat roughly as often as a random function (assuming that P_1 is a random permutation). Indeed previous works [19,8] also use repetitions in the values of $(y \oplus P_1(y))$ in their attacks.

Improved Attack from Multi-collisions. We can actually improve this attack and obtain an attack equivalent to the previous works from [19,8], by using the special structure of the 3-XOR instance (3). After building a partial multi-collision L_1 with $\Theta(n/\ln(n))$ values of f_1 starting with α, we look for pairs with $\big(f_0(x) \oplus f_2(z)\big)_{[0:n]} = \alpha$. Because of the structure of f_0 and f_2, we can just use $z = x \oplus \alpha$ for each known plaintext x. Therefore we have $|L_0| = |L_2|$ pairs partially colliding to a predefined value. Each couple gives a full collision if the second n-bit part corresponds to one of the elements in L_1; this happens with probability $n/\ln(n) \cdot 2^{-n}$. Thus this attack requires lists of size $D = Q = \mathcal{O}\big(2^n/(n/\ln(n))\big)$ in order to succeed with high probability in the KPA model.

We see that because we can choose parts of the inputs our problem may be easier than the purely random 3-XOR case. However generic algorithms are a good start to find dedicated cryptanalysis of 2EM. Moreover, the best known attacks against 2EM [19,8] can actually be reinterpreted in this way.

In this paper, we will give new attacks against 2EM starting from this 3-XOR presentation, and using algorithms based on the linear algebra approach.

3.3 A Key Recovery Algorithm

Now we describe a key recovery algorithm simply using the linear algebra 3-XOR algorithm by Joux [15] on the 3-XOR instance obtained by the reduction from 2EM. Using this algorithms as a black box, we have a time complexity of $\mathcal{O}(2^n/\sqrt{n})$ (since $w = 2n$). This is not as good as the best known 2EM key recovery, but this will lay the ground for the more efficient algorithms in Section 4.

The full attack can be written as Algorithm **GA**:

GA1. Compute $f_1(y) = (y \oplus P_1(y)) \| \; y$ for Q different values y and store them in L_1.

GA2. Compute $f_2(z) = z \parallel P_2(z)$ for Q different values z and store them in L_2.

GA3. Observe and find a set of n pairs of plaintext/ciphertext $(x, E(x))$ such that all $\{f_0(x) = x \parallel (x \oplus E(x))\}$ are linearly independent and store $x \parallel (x \oplus E(x))$ in L_0.

GA4. See L_0 as a $n \times 2n$ matrix. Use column reduction to find a $2n \times 2n$ transformation matrix M s.t. $L_0 M = [0_{n\times n} \parallel I_n]$.[2]

GA5. Right-multiply the lists with the transformation matrix:
$L'_0 \leftarrow L_0 M$; $L'_1 \leftarrow L_1 M$; $L'_2 \leftarrow L_2 M$.

GA6. Sort and find partial collisions in L'_1 and L'_2 on the first $n-$bit half. For each partial collisions $L'_1[i] \oplus L'_2[j]$ check whether the second n-bit half differs only on the h^{th} bit for some h. If yes go to **GA7**. If no solution found, algorithm fails.

GA7. A solution to the 3-XOR problem $(L_0[h], L_1[i], L_2[j])$ has been found. Output $k = x \oplus y$ with x the first half of $L_0[h]$ and y the second half of $L_1[i]$.

The main idea is that, since the transformation matrix M is linear, solving the 3-XOR problem for L'_0, L'_1, L'_2 yields the same solutions as L_0, L_1, L_2. Using the transformed lists is easier as we exploit the fact that $L'_0 = [0_{n\times n} \parallel I_{n\times n}]$ which is always possible to ensure after step **GA3**.

Step **GA3** will cost only n queries as n random words of size $2n$ will be linearly independent with very high probability. Note that because we just need to observe these, this attack works in the KPA setting.

Analysis. The query complexity Q is also the size of the lists L_1 and L_2. There are Q^2 pairs each XORing to one of the n elements of L_0 with probability $n/2^{2n}$ as they are taken randomly. Thus the probability of step **GA6** succeeding is $(n \cdot Q^2)/2^{2n}$.

Therefore for a constant success probability we fix $(n \cdot Q^2)/2^{2n} = \Theta(1)$. This leads to the following complexities: $Q = \mathcal{O}(2^n/\sqrt{n})$, $T = \mathcal{O}(2^n/\sqrt{n})$ and $D = \mathcal{O}(n)$.

We recall here that sorting random values and performing a right matrix multiplication $L_1 M$ (resp. $L_2 M$) on an exponentially large L_i are both computed in time linear with the size of L_i [4]. As for the computation of M, it is of polynomial time in n and therefore negligible.

Q is the query complexity and we find the relation $DQ^2 = 2^{2n}$ as expected. Memory-wise we need to store the full lists L_1 and L_2 so the memory complexity will also be $Q = \mathcal{O}(2^n/\sqrt{n})$.

Steps **GA1** and **GA2** concentrate all the permutation's evaluations but can be done as a pre-processing step.

[2] We write $L_0 = \begin{bmatrix} A & B \end{bmatrix}$. If B is non-singular, we can use $M = \begin{bmatrix} I & 0 \\ B^{-1}A & B^{-1} \end{bmatrix}$

4 Improved Attacks from the 3-XOR Problem

In the previous section we saw how tools to solve the 3-XOR problem could prove very useful for the 2EM key recovery attacks. But the cryptanalysis allows us to do some tweaks and have better results than simply applying the generic solutions.

In this section we will first show how to add a simple filter to Algorithm **GA** to mount an attack following the trade-off curve $DQ^2 = 2^{2n}$ while improving the time complexity of $T = 2^n/n$ (matching the best known 2EM attacks) and memory not exceeding Q. We also show how using the same filter but with the BDP algorithm adapted for the 3-XOR can give the best asymptotic time complexity so far, $T = \mathcal{O}(\frac{2^n \cdot \ln^2(n)}{n^2})$, though that largely remains theoretical.

Then we describe a very low-data and low-memory key recovery attack that essentially tweaks the previous Algorithm **GA** to a version that uses, for some parameter $0 < \lambda < 1$, few queries, $D = \lambda n$, time $Q = T = 2^n/\lambda n$ and memory $2^{\lambda n}$. This actually beats the best information theoretic distinguisher known so far in this range of very low data ($DQ^2 < 2^{2n}$).

4.1 Clamping to a Smaller 3-XOR Instance

We first describe an efficient algorithm with a large trade-off space with parameter $D = |L_0| = 2^d$ and $Q = |L_1| = |L_2| = 2^{n-d/2}$ and time complexity $\mathcal{O}(2^n/n)$ (independently of D and Q). This algorithm is built from the 3-XOR algorithm of [4], but we take advantage of the structure of the 3-XOR problem to reduce the time complexity below $\mathcal{O}(2^n/\sqrt{n})$ (reached by Algorithm GA). Indeed, our 3-XOR instance is given as:

$$\begin{aligned} f_0(x) &:= x && \| \; x \oplus E(x) \\ f_1(y) &:= y \oplus P_1(y) \| \; y \\ f_2(z) &:= z && \| \; P_2(z) \end{aligned} \tag{3}$$

We can use a variant of the clamping trick of Bernstein [2] to simplify this instance. For a parameter d, we consider the $2^{n-d/2}$ values y with $y_{[0:d/2]} = 0$ and we evaluate f_1 on those values. This gives a list L_1 with $|L_1| = 2^{n-d/2}$ such that all values have $d/2$ zero bits ($L_1[i]_{[n:n+d/2]} = 0$). Similarly, we consider all values z' with $z'_{[0:d/2]} = 0$, and we evaluate f_2 on $z = P_2^{-1}(z')$ to build a list L_2 with $L_2[j]_{[n:n+d/2]} = 0$. Finally, we consider 2^d known plaintexts x, and we keep the values with $\big(x \oplus E(x)\big)_{[0:d/2]} = 0$ in a list L_0. We expect to have $|L_0| = 2^{d/2}$. We now have three lists with $L_i[u]_{[n:n+d/2]} = 0$, so we can consider this as a 3-XOR problem on $w = 2n - d/2$ bits. We have $|L_0| \cdot |L_1| \cdot |L_2| = 2^{2n-d/2} = 2^w$; therefore there is on average one solution, and the algorithm of Bouillaguet *et al.* [4] finds it with complexity $\mathcal{O}\big(|L_0| \cdot (|L_1| + |L_2|)/w\big) = \mathcal{O}(2^n/n)$.

When writing the full details, we have Algorithm **CL**:

CL1. Compute $f_1(y) = (y \oplus P_1(y)) \| \; y$ for all y such that $y_{[0:d/2]} = 0$. Remove bits $[n : n + d/2]$ (fixed to 0) and store the $(2n - d/2)$-bit values in L_1.

CL2. Compute $f_2(P_2^{-1}(z')) = P_2^{-1}(z') \parallel z'$ for all z' such that $z'_{[0:d/2]} = 0$. Remove bits $[n : n + d/2]$ (fixed to 0) and store the $(2n - d/2)$-bit values in L_2.

CL3. Until a solution is found do:

CL3.1. Capture and filter a set of n pairs of plaintext/ciphertext $(x, E(x))$ such that $(x \oplus E(x))_{[0:d/2]} = 0$ and all $\{f_0(x) = x \parallel (x \oplus E(x))\}$ are linearly independent. Remove bits $[n : n + d/2]$ (fixed to 0) and store the $(2n - d/2)$-bit values in L_0.

CL3.2. See L_0 as an $n \times (2n - d/2)$ matrix. Use column reduction to find the $(2n - d/2) \times (2n - d/2)$ transformation matrix M such that $L_0 M = [0_{n\times(n-d/2)} \parallel I_n]$.

CL3.3. Right-multiply the lists with the transformation matrix:
$L'_0 \leftarrow L_0 M$; $L'_1 \leftarrow L_1 M$; $L'_2 \leftarrow L_2 M$.

CL3.4. Sort and find partial collisions in L'_1 and L'_2 on the first $(n - d/2)$-bit prefix. For each partial collisions $L'_1[i] \oplus L'_2[j]$ check whether the second n-bit part differs only on the h^{th} bit for some h. If yes go to **CL4**. If no solution found, loop on **CL3**.

CL4. A solution to the 3-XOR problem $(L_0[h], L_1[i], L_2[j])$ has been found. Output $k = x \oplus y$ with x the first n-bit of $L_0[h]$ and y made of $d/2$ zeros followed with the last $n - d/2$ bits of $L_1[i]$.

In steps **CL1** and **CL2** we only fixed the $d/2$ first bits so that we have lists of size $2^{n-d/2}$. Step **CL2** still constructs the usual L_2 as a collection of $z \parallel P_2(z)$ only we need to fix the values of $P_2(z) = z'$ and compute the value $z = P_2^{-1}(z')$ using the inverse.

Then all of this works very much like Algorithm **GA** the main difference begin at step **CL3.1** where we filter the observed pairs. Indeed we look for a triplet such that $x \oplus y = z' \oplus E(x)$ so fixing bits of y and z' fixes bits of $(x \oplus E(x))$.

4.2 Complexity Analysis

Data Complexity. The data complexity depends on the number of plaintext/ciphertext pairs we will expect to observe before we find a solution. One way to see it is to count the number of observable right triplets. Initially there are 2^n right triplets but we restrict ourselves to triplets such that $y_{[0:d/2]} = 0$ and $P_2(z)_{[0:d/2]} = 0$, a d-bit filter, so on average will remain 2^{n-d} right triplets. Therefore the moment we observe an x belonging to one of these right triplets it will necessarily pass the filter, give a solution and finish the algorithm. This happens with probability $2^{n-d}/2^n = 2^{-d}$ therefore we expect solution after $D = 2^d$ pairs $(x, E(x))$.

Memory Complexity. The largest lists in memory are L_1 and L_2 that require, in the balanced case, $\mathcal{O}(2^{n-d/2})$ blocks of memory.

Query Complexity. The offline query complexity is also the size of L_1 and L_2, that is $2^{n-d/2} = Q$. In particular, we use as much data as the best known distinguisher with $D \cdot Q^2 = 2^n$. Notice that for the balanced case $D = Q = 2^{2n/3}$ this attack is optimal in the information theoretic model as Chen *et al.* [5] proved that $\mathcal{O}(2^{2n/3})$ is a lower bound.

Time Complexity. First we need to compute both lists L_1 and L_2 requiring to compute $2^{n-d/2}$ permutations each (this can be a precomputation). We expect the algorithm to succeed after 2^d pairs $(x, E(x))$ with good probability. Thanks to the $d/2$-bit filter in step **CL3.1** only $2^{d/2}$ pairs are expected to be processed by batches of n values. Therefore we expect to do $2^{d/2}/n$ loops **CL3** before we finish. Each loop consists of computing a small transformation matrix, applying it to the big lists L_1 and L_2, sorting them and looking for prefix collisions. All of these costs are linear in the lists size, $2^{n-d/2}$, or in the number of expected $(n-d/2)$-bit prefix collisions in **CL3.4** that is $|L_1| \cdot |L_2|/2^{n-d/2} = 2^{n-d/2}$. Therefore each loop costs $\mathcal{O}(2^{n-d/2})$ and is expected to be performed $2^{d/2}/n$ times for a total computational time complexity of $T = \mathcal{O}(2^n/n)$. This computational time is independent of d.

Discussion. Algorithm **CL** achieves a computational time complexity of $T = 2^n/n$ while using as much information as the best known information theoretic attack with $D \cdot Q^2 = 2^{2n}$. In particular this is information theoretically optimal in the balanced case $D = Q = 2^{2n/3}$ that is for $d = 2n/3$. This attack works with known plaintexts, and there is no obvious way to improve it using chosen plaintext.

For most of the choices of d, evaluations of the cipher and the permutations is not the dominant cost of the algorithm. In this analysis we assume that operations on n-bit words and memory access to lists L_1 and L_2 cost $\theta(1)$ evaluations of the cipher, but if we assume instead that they cost much less than one evaluation (as done in [14]) the attack is even more interesting.

To optimize the memory complexity that is $2^{n-d/2}$, we need to choose a fairly high value d. In that case the data complexity $D = 2^d$ becomes problematic but we can swap the number of online call to E with the number of offline calls to P_1, effectively swapping f_0 and f_1, thanks to the symmetry highlighted in Section 2.3. This gives a data and memory complexity of $2^{n-d/2}$, a query complexity of $Q = 2^{n-d/2-1} + 2^{d-1}$ and the time remains $T = \mathcal{O}(2^n/n)$. This becomes a Chosen Plaintext attack because step **CL1** requires to choose part of inputs. Concrete values for $n = 64$ for such trade-off are given in Table 2 as "optim. memory & swap $E \leftrightarrow P_1$".

4.3 Using Baran-Demaine-Pătraşcu's 3-SUM Algorithm

Since the previous algorithm just uses a 3-XOR algorithm as a black box after clamping, we can also use it with the BDP algorithm adapted to 3-XOR [4]. In fact, any 3-XOR algorithm could be used after clamping which implies that an improved random 3-XOR algorithm would lead to an improved 2EM cryptanalysis.

This adapted BDP algorithm has a better asymptotic complexity, with a speed-up of $\frac{w^2}{\ln^2(w)}$ compared to the quadratic algorithm.

This results in a key-recovery attack against 2EM with asymptotic time complexity $\mathcal{O}(2^n \cdot \ln^2(n)/n^2)$. This is asymptotically better than the best known 2EM key recoveries. However, as shown in [4], it is not practical for realistic word sizes w. Indeed the dominant term in the complexity of the BDP algorithm is $\mathcal{O}(|L_0|.|L_1|/m^2)$ with $m = \Theta(n/\ln(n))$. Following the analysis of Bouillaguet *et al.*, we have more concretely $m \simeq n/(112\ln(n))$. Therefore, in order to have $m^2 > n$, we would need $n > 2.75 \times 10^6$.

4.4 Very Low Data Algorithm

The previous Algorithm **CL** can reach a low-data complexity (with a small parameter d) that would be a multiple of n, or a relatively low-memory complexity (close to $2^{n/2}$ with a large d), and having both close to $2^{n/2}$ requires chosen plaintexts. We now describe a new algorithm that combines a very low-data complexity and a low memory. This algorithm uses only $D = \lambda n$ known plaintexts for $0 < \lambda < 1$, and has a time complexity $T = \mathcal{O}(2^n/\lambda n)$ while using only a memory of size $2^{\lambda n}$. Moreover, we have $D \cdot Q = 2^n$ and $D \cdot Q^2 = \mathcal{O}(2^{2n}/\lambda n)$, that is the best information theoretical trade-off so far between online and offline queries.

This will be algorithm **LD** with parameter $0 < \lambda < 1$ (typically, we have $\lambda = 1/2$):

LD1. Observe and find a set of λn pairs of plaintext/ciphertext $(x, E(x))$ such that all $\{(x \oplus E(x))_{[n-\lambda n:n]}\}$ are linearly independent and store $f_0(x) = x \,\|\, (x \oplus E(x))$ in L_0.

LD2. See L_0 as a three concatenated λn-line matrices:

$$L_0 = [\underbrace{A}_{n} \| \underbrace{B}_{n-\lambda n} \| \underbrace{C}_{\lambda n}]$$

Define the $n \times n$ small transformation matrix M_s:

$$M_s = \begin{bmatrix} I & 0 \\ C^{-1}B & C^{-1} \end{bmatrix} \qquad M_s^{-1} = \begin{bmatrix} I & 0 \\ B & C \end{bmatrix}$$

and the $2n \times 2n$ big transformation matrix M:

$$M = \begin{bmatrix} I & 0 \\ \left(M_s \begin{bmatrix} 0 \\ A \end{bmatrix}\right) & M_s \end{bmatrix} = \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ C^{-1}A & C^{-1}B & C^{-1} \end{bmatrix}$$

LD3. Right-multiply the list L_0 with the big transformation matrix: $L_0' \leftarrow L_0 M = [\underbrace{0}_{n} \| \underbrace{0}_{n-\lambda n} \| \underbrace{I}_{\lambda n}]$

LD4. Until a solution is found pick a new $(n - \lambda n)$-bit value α and do:

LD4.1. For all λn-bit value u compute $f_1\big([\alpha|u]\cdot M_s^{-1}\big) = [\alpha|u]\cdot M_s^{-1} \oplus P_1\big([\alpha|u]\cdot M_s^{-1}\big) \parallel [\alpha|u]\cdot M_s^{-1}$. Store them in L_1.

LD4.2. For all λn-bit value u compute $f_2\big(P_2^{-1}([\alpha|u]\cdot M_s^{-1})\big) = P_2^{-1}([\alpha|u]\cdot M_s^{-1}) \parallel [\alpha|u]\cdot M_s^{-1}$. Store them in L_2

LD4.3. Modify the lists with the big transformation matrix:
$L_1' \leftarrow L_1 M;\quad L_2' \leftarrow L_2 M$.
Note that all elements of L_1' and L_2' have bits $[n : n+\lambda n]$ set to α.

LD4.4. Sort and find partial collisions in L_1' and L_2' on the first n-bit half. For each partial collisions $L_1'[i] \oplus L_2'[j]$ check whether the second half differs on a single bit h with $n - \lambda n < h \leq n$. If yes go to **LD5**. If no solution found, continue to loop on **LD4**.

LD5. A solution to the 3-XOR problem $(L_0[h-(n-\lambda n)], L_1[i], L_2[j])$ has been found. Output $k = x \oplus y$ with x the first half of $L_0[h-(n-\lambda n)]$ and y the second half of $L_1[i]$.

We again use the property that finding a solution for the 3-XOR in the modified lists yield the same solution in the original lists.

With the way we defined the big transformation matrix M in **LD2** and the fact that we applied M_s^{-1} to the inputs in steps **LD4.1** and **LD4.2**, when we perform step **LD4.3** we get the values $f_1\big([\alpha|u]\cdot M_s^{-1}\big)\cdot M = [\alpha|u]\cdot M_s^{-1} \oplus P_1\big([\alpha|u]\cdot M_s^{-1}\big) \oplus (0|(u\cdot A))\|[\alpha|u]$ and $f_2\big(P_2^{-1}([\alpha|u]\cdot M_s^{-1})\big)\cdot M = P_2^{-1}([\alpha|u]\cdot M_s^{-1}) \oplus (0|(u\cdot A))\|[\alpha|u]$ stored in L_1' and L_2' respectively. Thus the right-hand side of both lists reverts to the form $\{\alpha|u\}$ with fixed α and for all u. Therefore we get an $(n-\lambda n)$-bit collision for free on α matching with zeroes in L_0'.

4.5 Complexity Analysis

For this attack, in each loop we pick a value α and build L_1, L_2 of size $2^{\lambda n}$. Then we have a solution among the $2^{2\lambda n}$ pairs if one of them XORs to one of the λn values of L_0. Since we have a collision on $(n-\lambda n)$-bit value α for free, one couple gives a solution with probability $\lambda n \cdot 2^{-(n+\lambda n)}$. Thus each loop gives a solution with probability $2^{2\lambda n}\cdot \lambda n \cdot 2^{-(n+\lambda n)} = \lambda n \cdot 2^{\lambda n - n}$. For a constant probability of success we will need to perform around $\frac{2^{n-\lambda n}}{\lambda n}$ iterations.

Data Complexity. Step **LD1** completely determines the data complexity of the algorithm. We capture λn plaintext/ciphertext pairs and we get a linearly independent set of values with good probability. Therefore $D = \lambda n$ is the data complexity.

Memory Complexity. The list L_0 and the matrices take a space polynomial in n and therefore negligible. The lists L_1 and L_2 are always of size $2^{\lambda n}$. Therefore the memory complexity is $\mathcal{O}(2^{\lambda n})$.

Query Complexity. The computation of the public permutations are all done in steps **LD4.1** and **LD4.2** to build lists of size $2^{\lambda n}$. We pass through this step at each loop meaning that the total offline query complexity is:

$$Q = 2^{\lambda n} \cdot \frac{2^{n-\lambda n}}{\lambda n} = \frac{2^n}{\lambda n}$$

Time Complexity. Again, computations of the matrices in step **LD2** are essentially polynomial in n so negligible. Step **LD4.3** performs right-multiplications on large matrices and step **LD4.4** is about sorting and merging which makes those steps linear given that the merged list is of reasonable size. Here we have a partial collision on n bits with probability 2^{-n} therefore there will be around $2^{2\lambda n} \cdot 2^{-n} = 2^{2\lambda n - n}$ partial collisions that is less than the size of the lists ($2^{\lambda n}$) therefore step **LD4.4** has also a linear cost. The computational time complexity is therefore also led by the query complexity that is $T = \frac{2^n}{\lambda n}$.

Acceptable Range. Notice that the previous reasoning to derive the time complexity is only applicable when we do need more than one loop to finish the algorithm as it makes no sense to multiply by half-a-round. So all those trade-off depending on λ are constraints by:

$$\frac{2^{n-\lambda n}}{\lambda n} \geq 1 \Leftrightarrow \lambda \leq \frac{W(2^n \ln 2)}{n \ln 2} = 1 - \frac{\ln(n \ln 2)}{n \ln 2} + o(1)$$

(using the Lambert W function)

Discussion. This attack works in the KPA setting as we only need to observe pairs of plaintext/ciphertext, and we need to observe surprisingly few of them, λn pairs are sufficient.

The memory requirement, $\mathcal{O}(2^{\lambda n})$, can also go quite low as we choose the parameter λ but this comes at the cost of no pre-computation possible as we need the transformation matrix to get the right inputs to the public permutations.

The computational time complexity $T = 2^n/\lambda n$ compares well with previous cryptanalysis done on this subject. So far there were no key recovery attack on 2EM with a better asymptotic complexity than $\mathcal{O}(2^n/n)$.

In the information theoretic model, trade-off between D and Q is important as a designer can always arbitrarily limit the maximum value of D by, for example, rekeying in order to achieve a certain security goal. In this regard, this algorithm has a better trade-off between the data and query complexity than the best known generic distinguisher by Gaži [13] that has the trade-off $DQ^2 = 2^{2n}$. Here $DQ^2 = 2^{2n}/\lambda n$ thus being the best known key recovery, and also the best distinguisher, for the acceptable range of λn.

In fact the proof by Chen *et al.* [5] says nothing for low-data range $D \leq 2^{n/4}$ and the best proof is therefore inherited from the original one round Even-Mansour scheme that lower-bounds the trade-off with $DQ \geq 2^n$. Gap between the best known distinguisher and the proof in this range is still an open problem but Algorithm **LD**, which has the trade-off $DQ = 2^n$ —and also $DT = 2^n$ —for any λ, proves for the first time the optimality of the original proof of the trade-off between D and Q for the acceptable range of λ that is for $1 \leq D \leq \frac{W(2^n \ln 2)}{\ln 2}$.

Previous Work. We can see this cryptanalysis as an advanced version of the attack by Dinur *et al.* using linear algebra [9, Section 4.2]. We can list three main differences that make this attack an improvement over the previous one. First, as already mentioned in Section 2.3, we use the symmetry between E, P_1, P_2 to reduce the data complexity from $2^n/\lambda n$ to λn. Then the use of the big transformation matrix M, that essentially performs a Gaussian elimination over the whole $2n$-bit words, makes the attack works with known plaintexts while Dinur *et al.* required chosen plaintexts (even after applying the symmetry trick). Finally, the resulting n-bit filter of step **LD4.4** allows for a larger acceptable range of λ than the previous attack that needed $\lambda < 1/3$ to limit the number of partial collisions.

5 Extension to r rounds.

The approach can be generalized to attack multiple rounds. In fact the cryptanalysis of a single key r-round EM scheme can be written as a $(r+1)$-XOR problem with words of size rn. Even though for $r \geq 4$ generic algorithms won't directly provide interesting attacks with competitive computational complexity, this elegantly rewrites the known generic distinguisher on rEM and may be a good start to look for dedicated cryptanalysis.

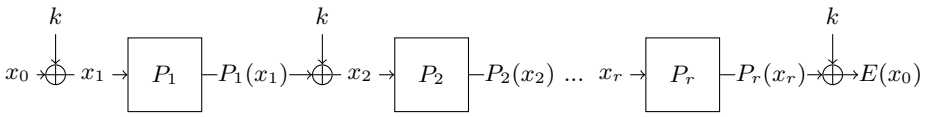

Fig. 5. A right tuple gives a path of rEM

Definition 3 (k-XOR problem). *Given k functions $f_0, f_1, f_2, ..., f_k$, find k inputs $(x_0, x_1, x_2, ..., x_k)$ such that $f_0(x_0) \oplus f_1(x_1) \oplus f_2(x_2) \oplus ... \oplus f_k(x_k) = 0$.*

Extended Relation. To see that we follow the same reasoning as in Section 2.2 but for the r-round EM, Figure 5, and look for an $(r+1)$-tuple $(x_0, x_1, ..., x_r)$ satisfying the generalized relation $\mathcal{R}$:

$$\mathcal{R}(x_0, x_1, x_2, ..., x_r) := \begin{cases} x_0 \oplus x_1 = k \\ P_i(x_i) \oplus x_{i+1} = k, \quad 1 \leq i \leq r-1 \\ P_r(x_r) \oplus E(x_0) = k \end{cases} \tag{4}$$

$$\Rightarrow \begin{cases} x_0 \oplus x_1 = P_1(x_1) \oplus x_2 \\ P_i(x_i) \oplus x_{i+1} = P_{i+1}(x_{i+1}) \oplus x_{i+2}, \quad 1 \leq i \leq r-2 \\ P_{r-1}(x_{r-1}) \oplus x_r = P_r(x_r) \oplus E(x_0) \end{cases} \tag{5}$$

Again we cannot directly observe $\mathcal{R}$ but we can observe the implied relation 5 which is an rn-bit filter and is enough so that a random $(r+1)$-tuple satisfying Filter 5 is a right tuple with good probability.

Define Lists. Now we can define $r+1$ lists of r n-bit entries such that solving the $(r+1)$-XOR problem on those lists over all entries trivially gives a solution to 5:

$$L_0[h] := \begin{cases} x_0 & ,h=1 \\ 0 & ,2 \le h \le r-1 \\ E(x_0) & ,h=r \end{cases} \quad L_1[h] := \begin{cases} x_1 \oplus P_1(x_1) & ,h=1 \\ P_1(x_1) & ,h=2 \\ 0 & ,h \ge 3 \end{cases}$$

$$\underset{2\le i\le r-1}{L_i[h]} := \begin{cases} 0 & ,h \le i-2 \\ x_i & ,h=i-1 \\ x_i \oplus P_i(x_i) & ,h=i \\ P_i(x_i) & ,h=i+1 \\ 0 & ,h \ge i+2 \end{cases} \quad L_r[h] := \begin{cases} 0 & ,h \le r-2 \\ x_r & ,h=r-1 \\ x_r \oplus P_r(x_r) & ,h=r \end{cases}$$

see example for $r=5$ in Table 3. Thus this indeed defines an $(r+1)$-XOR problem with rn-bit words even though it is more structured than the purely random k-XOR problem. Upon its resolution we have a successful key recovery with good probability when guessing $k = x_0 \oplus x_1$.

Table 3. Cryptanalysis of 5EM.

Lists' construction for a cryptanalysis using the 6-XOR problem.					
$L_0 \ni \{$	x_0	.	.	.	$E(x_0)\}$
$L_1 \ni \{$	$x_1 \oplus P_1(x_1)$	$P_1(x_1)$	.	.	. $\}$
$L_2 \ni \{$	x_2	$x_2 \oplus P_2(x_2)$	$P_2(x_2)$	.	. $\}$
$L_3 \ni \{$	.	x_3	$x_3 \oplus P_3(x_3)$	$P_3(x_3)$	. $\}$
$L_4 \ni \{$	.	.	x_4	$x_4 \oplus P_4(x_4)$	$P_4(x_4)\}$
$L_5 \ni \{$	.	.	.	x_5	$x_5 \oplus P_5(x_5)\}$

Generic Cryptanalysis. Even though the problem is structured this allows us to use generic algorithms for the k-XOR problem to perform a cryptanalysis. With purely random functions it is known that the lower bounds of queries for the the rn-bit words $(r+1)$-XOR problem is $\mathcal{O}(2^{\frac{rn}{r+1}})$. Interestingly this exactly coincides with the lower bound queries for the single key r-round Even-Mansour scheme [3]. Using generic algorithms allows a cryptanalysis using $D = Q = \mathcal{O}(2^{\frac{rn}{r+1}})$ therefore

being optimal in query complexity. In fact the approach can be thought as similar to the simple known distinguisher but instead of looking for contradictory paths we directly look for a correct path (that implies a right tuple) and guess the key.

Limitation. The computational time complexity of generic algorithms by Wagner for this problem is $T = \mathcal{O}\left(r \cdot 2^{\frac{rn}{\lfloor \log(r+1) \rfloor + 1}}\right)$ [20]. For $r = 2$ and 3 rounds this is just $\mathcal{O}(2^n)$ and we could improve from there in the 2EM case. For the 3EM case Dinur *et al.* [9] showed that we can have a complexity below $\mathcal{O}(2^n)$ using multicollisions and while it is fairly straightforward to rewrite the same attack in the 4-XOR context it is also non-trivial to improve this.

On the other hand the complexity is way over 2^n for $r \geq 4$ rounds. Therefore this is mainly an information theoretic attack. However the lists here have a strong structure, see Table 3, with many bits to 0 which opens the question of a dedicated algorithm with competitive computational time/memory trade-off.

6 Conclusion

In this paper we presented a 3-XOR approach to key-recovery attacks on single-key two-round Even-Mansour. That allows us to gain a better understanding of previous works and devise competitive algorithms using linear algebra techniques that have been initially developed for the random 3-XOR problem.

These attacks have a particularly interesting data and memory complexities. In particular, we give the first attacks where both the data and memory complexity are below $\mathcal{O}(2^{n-\varepsilon})$ for $\varepsilon > 0$, while achieving the best known time complexity of $\mathcal{O}(2^n/n)$. Previous attacks with a similar time complexity required either a very large memory or very large data, making them unlikely to be useful in practice. We also give an attack that improves the asymptotic time complexity to $\mathcal{O}(2^n \cdot \ln^2(n)/n^2)$, although it is not applicable for practical values of n. As another interesting result, we show a very low-data attack that beats the best known distinguisher, and actually matches the proven lower bound for single round Even-Mansour construction, with $DT = 2^n$.

All those attacks are shown on the 2EM construction with no key schedule and independent permutations, but we prove that an attack on this variant of 2EM leads to an attack on the more general 2EM with a linear key schedule. Additionally we show that the 2EM construction has an implicit symmetry that allows to blindly swap the number of calls one makes to each oracle during an attack; this automatically allows new trade-offs between the parameters.

Iterated Even-Mansour schemes are idealized SPN networks and understanding their security is important because many block ciphers, including the AES, are based on this design. In this work we focused on the two-round construction linking it to the 3-XOR problem such that a future improvement of the random 3-XOR algorithms will improve our cryptanalysis. But we can also extend this approach to r-round constructions and the $(r+1)$-XOR problem with a particular structure. We detail this link in Section 5 but additional work is required to build competitive key-recovery attacks from that.

Acknowledgement

Part of this work was supported by the French DGA.

References

1. Baran, I., Demaine, E.D., Pătraşcu, M.: Subquadratic algorithms for 3sum. Algorithmica 50(4), 584–596 (Apr 2008), https://doi.org/10.1007/s00453-007-9036-3
2. Bernstein, D.J.: Better price-performance ratios for generalized birthday attacks. In: Workshop Record of SHARCS. vol. 7, p. 160 (2007)
3. Bogdanov, A., Knudsen, L.R., Leander, G., Standaert, F.X., Steinberger, J.P., Tischhauser, E.: Key-alternating ciphers in a provable setting: Encryption using a small number of public permutations - (extended abstract). In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 45–62. Springer, Heidelberg (Apr 2012)
4. Bouillaguet, C., Delaplace, C., Fouque, P.A.: Revisiting and improving algorithms for the 3xor problem. IACR Trans. Symm. Cryptol. 2018(1), 254–276 (2018)
5. Chen, S., Lampe, R., Lee, J., Seurin, Y., Steinberger, J.P.: Minimizing the two-round Even-Mansour cipher. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part I. LNCS, vol. 8616, pp. 39–56. Springer, Heidelberg (Aug 2014)
6. Chen, S., Steinberger, J.P.: Tight security bounds for key-alternating ciphers. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 327–350. Springer, Heidelberg (May 2014)
7. Daemen, J.: Limitations of the Even-Mansour construction (rump session). In: Imai, H., Rivest, R.L., Matsumoto, T. (eds.) ASIACRYPT'91. LNCS, vol. 739, pp. 495–498. Springer, Heidelberg (Nov 1993)
8. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Key recovery attacks on 3-round Even-Mansour, 8-step LED-128, and full AES2. In: Sako, K., Sarkar, P. (eds.) ASIACRYPT 2013, Part I. LNCS, vol. 8269, pp. 337–356. Springer, Heidelberg (Dec 2013)
9. Dinur, I., Dunkelman, O., Keller, N., Shamir, A.: Key recovery attacks on iterated Even-Mansour encryption schemes. Journal of Cryptology 29(4), 697–728 (Oct 2016)
10. Dunkelman, O., Keller, N., Shamir, A.: Minimalism in cryptography: The Even-Mansour scheme revisited. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 336–354. Springer, Heidelberg (Apr 2012)
11. Dunkelman, O., Keller, N., Shamir, A.: Slidex attacks on the Even-Mansour encryption scheme. Journal of Cryptology 28(1), 1–28 (Jan 2015)
12. Even, S., Mansour, Y.: A construction of a cipher from a single pseudorandom permutation. In: Imai, H., Rivest, R.L., Matsumoto, T. (eds.) ASIACRYPT'91. LNCS, vol. 739, pp. 210–224. Springer, Heidelberg (Nov 1993)
13. Gaži, P.: Plain versus randomized cascading-based key-length extension for block ciphers. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013, Part I. LNCS, vol. 8042, pp. 551–570. Springer, Heidelberg (Aug 2013)
14. Isobe, T., Shibutani, K.: New key recovery attacks on minimal two-round Even-Mansour ciphers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017, Part I. LNCS, vol. 10624, pp. 244–263. Springer, Heidelberg (Dec 2017)
15. Joux, A.: Algorithmic Cryptanalysis. Chapman & Hall/CRC, 1st edn. (2009)

16. Joux, A., Lucks, S.: Improved generic algorithms for 3-collisions. In: Matsui, M. (ed.) ASIACRYPT 2009. LNCS, vol. 5912, pp. 347–363. Springer, Heidelberg (Dec 2009)
17. Lampe, R., Patarin, J., Seurin, Y.: An asymptotically tight security analysis of the iterated Even-Mansour cipher. In: Wang, X., Sako, K. (eds.) ASIACRYPT 2012. LNCS, vol. 7658, pp. 278–295. Springer, Heidelberg (Dec 2012)
18. Nikolic, I., Sasaki, Y.: Refinements of the k-tree algorithm for the generalized birthday problem. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015, Part II. LNCS, vol. 9453, pp. 683–703. Springer, Heidelberg (Nov / Dec 2015)
19. Nikolic, I., Wang, L., Wu, S.: Cryptanalysis of round-reduced LED. In: Moriai, S. (ed.) FSE 2013. LNCS, vol. 8424, pp. 112–129. Springer, Heidelberg (Mar 2014)
20. Wagner, D.: A generalized birthday problem. In: Yung, M. (ed.) CRYPTO 2002. LNCS, vol. 2442, pp. 288–303. Springer, Heidelberg (Aug 2002)